**PCT** WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 7 : G06F 17/60 — A1

(11) International Publication Number: **WO 00/54191**

(43) International Publication Date: 14 September 2000 (14.09.00)

(21) International Application Number: PCT/CN99/00031

(22) International Filing Date: 8 March 1999 (08.03.99)

(71)(72) Applicant and Inventor: BI, Fujun [CN/CN]; 8/F Harmony Tower, 138 Wangfujing Street, Beijing 100006 (CN).

(72) Inventors; and

(75) Inventors/Applicants *(for US only)*: KOLT, Alexander, Joseph [US/US]; 124 Nassau ct, San Ramon, CA 9458 (US). COPPENS, William, E. [US/US]; 2005 Star Pine Way, Sanleandro, CA 94577 (US). BABBIDGE, John, Kelbe [AU/US]; Apartment A, 420 Cola Ballena, Alameda, CA 94501 (US). ZHOU, Gary [CN/US]; 172 Ratto Road, Alameda, CA 94502 (US). YAN, Hong [CN/US]; 268 East Ridge Drive, San Ramon, CA 94583 (US). BLISS, Shaun [GB/US]; 915 Shorepoint Court #E215, Alameda, CA 94501 (US).

(74) Agent: LIU, SHEN & ASSOCIATES; Huibin Building, A0601, No.8 Beichen Dong Street, Chaoyang District, Beijing 100101 (CN).

(81) Designated States: AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CU, CZ, DE, DK, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, UA, UG, US, UZ, VN, YU, ZW, ARIPO patent (GH, GM, KE, LS, MW, SD, SL, SZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

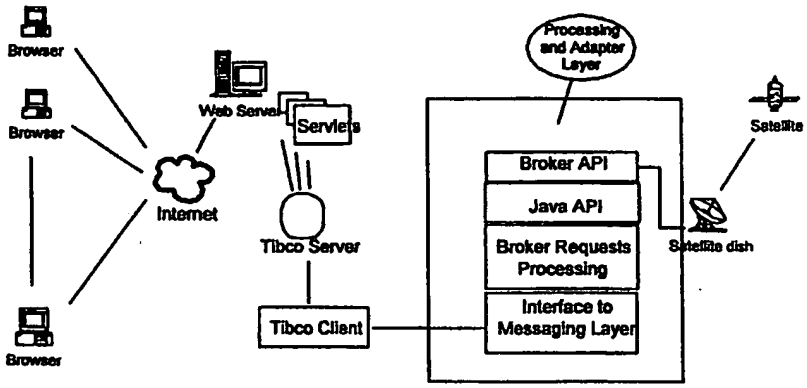
**Published**

*With international search report.*

(54) Title: A MULTI-BROKER CONNECTIVITY SYSTEM, AN ONLINE TRADING SYSTEM UTILIZING THE SAME, A MULTI-PROCESSING-SYSTEM NETWORKING SYSTEM, AND THE METHODS THEREFOR

(57) Abstract

Disclosed is a multi-broker connectivity system for networking to a plurality of broker systems, said connectivity system having a plurality of users, and comprising: a receiving means for receiving users' request and formatting the request into a predetermined format, said format including sender's ID and receiver's ID; a messaging means for transmitting data in said predetermined format to a predetermined receiver according to said receiver's ID; at least one processing/adapting means each adaptively interfaced to a specific broker system, for receiving request for the respective specific broker system so as to perform processing of a certain class on a received request, and returning the processing result in the same format as the corresponding request to said messaging means while roles of the sender and receiver are exchanged; a presenting means for receiving said returned result and de-formatting said result to present the result to the corresponding request sender. With such a system, as Universal Broker Connectivity (UBC) Messaging Architecture is achieved, which will produce business functionality at the front-end independent of backend technology constraints and have a middle tier completely generic and useful to all functionality.




EXPLORATEUR
SERVEUR WEB
MINISERVEUR
SERVEUR TIBCO
CLIENT TIBCO
COUCHE DE TRAITEMENT/ADAPTATION
COURTIER API
TRAITEMENT DES DEMANDES DE COURTIER
COUPLAGE À LA COUCHE DE MESSAGERIE

*FOR THE PURPOSES OF INFORMATION ONLY*

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| AL | Albania | ES | Spain | LS | Lesotho | SI | Slovenia |
| AM | Armenia | FI | Finland | LT | Lithuania | SK | Slovakia |
| AT | Austria | FR | France | LU | Luxembourg | SN | Senegal |
| AU | Australia | GA | Gabon | LV | Latvia | SZ | Swaziland |
| AZ | Azerbaijan | GB | United Kingdom | MC | Monaco | TD | Chad |
| BA | Bosnia and Herzegovina | GE | Georgia | MD | Republic of Moldova | TG | Togo |
| BB | Barbados | GH | Ghana | MG | Madagascar | TJ | Tajikistan |
| BE | Belgium | GN | Guinea | MK | The former Yugoslav Republic of Macedonia | TM | Turkmenistan |
| BF | Burkina Faso | GR | Greece | | | TR | Turkey |
| BG | Bulgaria | HU | Hungary | ML | Mali | TT | Trinidad and Tobago |
| BJ | Benin | IE | Ireland | MN | Mongolia | UA | Ukraine |
| BR | Brazil | IL | Israel | MR | Mauritania | UG | Uganda |
| BY | Belarus | IS | Iceland | MW | Malawi | US | United States of America |
| CA | Canada | IT | Italy | MX | Mexico | UZ | Uzbekistan |
| CF | Central African Republic | JP | Japan | NE | Niger | VN | Viet Nam |
| CG | Congo | KE | Kenya | NL | Netherlands | YU | Yugoslavia |
| CH | Switzerland | KG | Kyrgyzstan | NO | Norway | ZW | Zimbabwe |
| CI | Côte d'Ivoire | KP | Democratic People's Republic of Korea | NZ | New Zealand | | |
| CM | Cameroon | | | PL | Poland | | |
| CN | China | KR | Republic of Korea | PT | Portugal | | |
| CU | Cuba | KZ | Kazakstan | RO | Romania | | |
| CZ | Czech Republic | LC | Saint Lucia | RU | Russian Federation | | |
| DE | Germany | LI | Liechtenstein | SD | Sudan | | |
| DK | Denmark | LK | Sri Lanka | SE | Sweden | | |
| EE | Estonia | LR | Liberia | SG | Singapore | | |

# A multi-broker connectivity system, an online trading system utilizing the same, a multi-processing-system networking system, and the methods therefor

## Field of the invention

The present invention relates to e-commerce technology, specifically, an internet-based multi-broker connectivity system, an online trading system utilizing the same, a multi-processing-system networking system, and the methodss therefor. Said multi-broker connectivity system or online trading system can be connected to a plurality of broke systems to enable investors to conduct securities online-trading via internet, and provide an infrastructure that will produce business functionality at the front-end independent of backend technology constraints and have a middle tier completely generic and useful to all functionality.

## Technical background

With the emergence of Internet technology and growth of the Internet application in the society, our society is becoming an information society. E-commerce based on information technology is used in many fields, such as ticket ordering, purchasing via Internet, securities trading, etc.

The securities market is always a concern for investors, and there is growing demand for tools to help the investors to conduct remote online trading, without going to the broker offices. There are many electronic securities trading system at the respective broker offices, such as LAN online trading system, card trading system, etc. There are already some online trading

systems based on Internet technology, with which a broker's end users can access to the web page of this broker and conduct online securities trading.

Fig. 1 illustrates the conventional approach to online trading systems. As shown in Fig. 1, each broker must have its own online trading system, including its own web server networking to the exchange via leased lines. The end users access to the broker system at the broker's web site and conduct online trading with this broker's system only. The broker must keep its own technical team to develop its online trading system, to maintain the system. It is not only costly to keep such a technical team, but difficult for the broker to provide further value added services on its online trading system, since each broker system has its own separate end users, and the number of end users of each broker system is limited.

There is a need for an online trading system capable of connecting multiple broker systems. However, even if one wants to establish such a system, he may meet a lot of outstanding problems. For example, because each of the broker systems may be developed by different technical teams, they may be of different kinds, and may have different programming, different hardware and software configuration, and have different data formats. For such a system, there are involved many end users and many broker systems, how to dispatch the related information to the destination receiver is still an open question. There are already some existing online trading systems of this kind, such as Stockstar in Shanghai, which can be networked to multiple broker systems, and the users can perform online trading via its web site: stockstar.com. However, its solution is not a real many(users) to many(brokers) solution, since from the users to the system, it is a many-to-

one, while from the system to the broker systems, it is in fact a one-to-one solution, that means each broker system is connected to the Stockstar system via a DDN leased line, which is costly. The data formats transmitted in these DDN lines depend on the specific broker system, and in the Stockstar system, it needs some format translation to convert the data format into TCP/IP that is utilized in the internet environment. Therefore, the essential part of the existing solution is still a many(users)-to-one(broker) solution, although it can be networked to a plurality of broker systems.

## Object of the invention

The object of the present invention is to provide a many(users)-to-many(brokers) solution, which is a sophisticated technical service layer between the end users and the broker systems, to provide online trading services of different broker systems to the end users. This technical service layer is not a broker system, but provides networking service to broker trading systems, and provides online trading functionality and value-added services to all the end users. This many-to-many solution is an infrastructure that will produce business functionality at the front-end independent of backend technology constraints and have a middle tier completely generic and useful to all functionality.

Specifically, one object of the present invention is to provide a multi-processing-system networking system for networking to a plurality of processing systems, throughwhich a user can be networked to a plurality of different kinds of broke trading processing systems to conduct bi-directional data transmissions and real-time processing on his demands, and the method thereof.

Another object of the present invention is to provide a multi-broker networking or universal broker connectivity system, which can be adaptively networked to a plurality of broke trading systems, to provide online trading services of different broker systems to the end users.

Still another object of the present invention is to provide an online trading system, which can be networked to a plurality of broker trading systems, so as to provide online trading services of different broker systems to the end users at a same web site.

Still another object of the present invention is to provide a method for adapting a common connectivity system to a plurality of processing systems, by which different processing systems can constitute a uniformed processing system with the common connectivity system to work as a whole.

## Summary of the invention

To achieve the above objects, the present invention provides a multi-processing-system networking system for networking to a plurality of processing systems, said networking system having a plurality of users, and comprising: a receiving means for receiving users' request and formatting the request into a predetermined format, said format including sender's ID and receiver's ID; a messaging means for transmitting data in said predetermined format to a predetermined receiver according to said receiver's ID; at least one processing/adapting means each adaptively interfaced to a specific processing system, for receiving request for the

respective specific processing system so as to perform processing of a certain class on a received request, and returning the processing result in the same format as the corresponding request to said messaging means while roles of the sender and receiver are exchanged; a presenting means for receiving said returned result and de-formatting said result to present the result to the corresponding request sender.

Further, the present invention provides a method for networking users to a plurality of processing systems, comprising the steps of: receiving user's request and formatting the request into a predetermined format, said format including sender's ID and receiver's ID; transmitting said request in said predetermined format to a predetermined receiver according to said receiver's ID; receiving said request in said predetermined format for the respective specific processing system and performing processing of a certain class on the received request; transmitting said processing result in the same format as the corresponding request from said receiver to said sender according to said sender's ID; de-formatting said return result to present the result to the corresponding request sender.

The present invention also provides a multi-broker connectivity system for networking to a plurality of broker systems, said connectivity system having a plurality of users, and comprising: a receiving means for receiving users' request and formatting the request into a predetermined format, said format including sender's ID and receiver's ID; a messaging means for transmitting data in said predetermined format to a predetermined receiver according to said receiver's ID; at least one processing/adapting means each adaptively interfaced to a specific broker system, for receiving request for the

respective specific broker system so as to perform processing of a certain class on a received request, and returning the processing result in the same format as the corresponding request to said messaging means while roles of the sender and receiver are exchanged; a presenting means for receiving said returned result and de-formatting said result to present the result to the corresponding request sender.

Moreover, the present invention further provides an online trading system utilizing the multi-broker-connectivity system described above, wherein said multi-broker-connectivity system is networked to each of the broker systems by means of VPN via internet.

Moreover, the present invention provides a method for adapting a common connectivity system to a plurality of processing systems, each has a native API including a plurality of processing functions, comprising the steps of: providing a uniformed adapting interface, a first API layer, a second API layer in sequence between the common connectivity system and each of the specific processing system; inputting uniformed processing instructions from the common connectivity system and to a specific processing system; calling respective functions of the first API layer according to the instruction's type; calling respective functions of the second API layer according to said respective functions of the first API layer; and calling respective processing functions of said native processing system's API according to said respective functions of the second API layer.

**Brief description of the drawings**

Fig. 1 illustrates the conventional online trading systems;

Fig.2a is an overview of arrangement of the online trading system to which the present invention is applied;

Fig.2b illustrates the layer structure of this invention formed between the end users and the brokerage;

Fig. 3 illustrates multi-layer architecture of an embodiment of the online trading system according to the present invention;

Fig. 4 illustrates further architecture of a multi-broker networking system that forms a vital part of the online trading system in Fig.3;

Fig. 5 illustrates the API architecture at broker side;

Fig. 6 illustrates the interface to messaging layer at a broker system side;

Fig. 7 illustrates the related processing involved in the multi-broker networking system;

Fig. 8 illustrates the processing involved in the servlet at the web server;

Fig. 9 illustrates physical architecture of an embodiment of the present invention, wherein quote and other news and other service can be provided utilizing the same TIBCO P/S virtual bus;

Fig. 10 illustrates a configuration for sending information to one of user pre-defined facilities, such as pager, fax or e-mail.

## Detailed description of the preferred embodiments

Fig.2a is an overview of arrangement of the online trading system to which the present invention is applied. As shown in Fig. 2a, the solution of this invention provides a web site, for example, www.99stock.com (or www.99stock.com.cn ), to be accessed by all the investors or end users via internet. The 99stock system is networked to a plurality of broker trading systems by means of universal broker connectivity (multi-broker

networking) technology, so that the 99stock system can provide online trading services of different broker systems to all the investors.

Fig.2b shows the layer structure between the end users and the brokerage. This invention provides a unique technical service layer, a many-to-many solution, between the end users and the brokerage. On one hand, the 99stock system provides universal broker connectivity technology to all the possible broker systems; on the other hand, it provides private data provision technology to all the end users, to provide further value added services besides the online trading functionality.

Fig 3 shows the multi-layer architecture of the multi-broker networking(or universal broker connectivity) system according to the present invention. As shown in Fig. 3, it has a presentation layer, a messaging layer and a processing and adapter layer. The presentation layer that resides in a Web server is responsible for formatting and presenting data to the user, that was returned from the broker and is Broker specified. The Messaging layer contains a plurality of Messaging Servlets and Servers, in which a TIBCO unit is implemented which will be described later. The messaging servlets pack requests and send them to TIBCO and invoke presentation layer when results are received. The messaging servers receive requests from TIBCO, provide a queuing mechanism to handle over flow of incoming requests, provide a multithreaded processing environment fed by the queue, and invoke the appropriate processing class in the Processing Layer according to the Broker and request type. The Processing and adapter layer is a Broker specific layer designed to perform any broker specific functionality and is implemented on a request type by request type basis. Moreover, it provides

access to native broker APIs and performs message translation to appropriate format for native broker API.

Below, the Presentation Layer will be described in detail. One function of the Presentation Layer is to create in Web Server many leading set of pages, such as that of an Online Trading section links to by default needs to be changed to a broker directory style page . It also produces an HTML page for the user's Web browser, which enable the Messaging Servlet to call a broker specific Java class to format the data and return an HTML stream. It should be noted that any changes in formatting of the data or HTML presentation require code changes, recompilation of the relevant classes and redeployment of those classes. Moreover, it is preferred to use Java-based server Pages (JSP) for formatting data and call JSP pages with the Messaging Servlet. The planned use of JSP requires the servlet to call the appropriate JSP class that produces the required page. The JSP class runs a supplied method to access keys within a Hashtable. The Java Object mechanism is designed to service a Java applet. It would simply pass back the Java Object to the client applet over the HTTP connectivity using it as regular socket connectivity.

In the Messaging layer of Fig 3, a TIBCO product is provided for the messaging sending/receiving protocol using publish subscribe technology. TIBCO provides a subject-based key for clients to publish or subscribe to. Essentially a hierarchy of subjects can be created to allow clients to subscribe to information relevant to only them. This segregation is achieved by making the following assumptions:

- All brokers have 1 or more branches.

- Every Broker / Branch is a separate entity and is assumed completely independent of other branches.

To support the messaging structure, a level for each broker and a level for each branch should be created, a request level is then created as

Example: TRADING.Broker1.Branch1.request
TRADING.Broker1.branch2.request
TRADING.Broker2.branch1.request
TRADING.Broker3.branch1.request

A system subject is also provided to allow administrative changes to be performed on a large scale. This will be in the form of "TRADING.<brokerID>.<branchID>.system". This could be used to update all branch configurations for a particular broker, for example: "TRADING.broker1.*.system".

The Messaging Servlet performs critical functions. One of the main functions of the Servlet is to receive input data from a user via a HTTP post. This may be created either via a regular web page, JSP page or an applet. The servlet packages the information into a Hashtable. For example, the input Hashtable as the following one will minimally have the following entries, such as Customer ID, Broker ID, Branch ID, and RequestType, etc.

| Key | Description | Valid Values |
|---|---|---|
| CustomerID | Investors login | Any alpha numeric string |
| Password | Investors password | Any alpha numeric string |
| BrokerID | Arbitrary String representing the current broker | Any alpha numeric string |
| BranchID | Arbitrary String representing the | Any alpha numeric |

| | | |
|---|---|---|
| | current branch of the current broker. | string |
| RequestType | String that matches a class name at the broker Server | Orders |

RequestType information will not be validated at this level as the servlet is intended to eventually serve a plurality of brokers who may have different request transaction sets. Once packaged, the object will be published to TIBCO. The brokerID and BranchID keys in the object will be used as the TIBCO subject in the following form:

TRADING.<brokerID>.<branchID>.request

Using TIBCO's request publishing mechanism, the servlet will publish the broker request and automatically generate a listener or inbox that subscribes to the response for only that request. This mechanism will deliver the result data back to the Messaging Servlet.

The broker response will be received in the same Hashtable that was sent as a request. The Hashtable will then pass through the presentation layer. The presentation layer will eventually handle three types of output: HTML, JSP and Java Object. For example, the output Hashtable such as one listed below contains the following entries, such as: Customer ID, BrokerID, Branch ID, RequestType, Report and Error, etc.

| Key | Description | Valid Values | Data Source |
|---|---|---|---|
| CustomerID | Investors login | Any alpha numeric string | Unchanged from request |
| Password | Investors password | Any alpha numeric string | Unchanged from request |
| BrokerID | Arbitrary String representing the current broker | Any alpha numeric string | Unchanged from request |

| | | | |
|---|---|---|---|
| BranchID | Arbitrary String representing the current branch of the current broker. | Any alpha numeric string | Unchanged from request |
| RequestType | String that matches a class name at the broker Server | Orders | Unchanged from request |
| Report | Vector of Hashtables representing each row of data | Vector. May be NULL where only single, unique Strings are returned a no a plurality of records. | Supplied by broker specific transaction classes |
| Error | String, with an error or error code in it. | String. May be NULL if no error state occurred. | Supplied by broker specific transaction classes |

The Messaging Server will be explained as below. It is comprised of the TIBCO client connectivity layer and the generic request type classes. All request type classes will implement a Transaction Java interface that takes a Hashtable and a Broker object as input. The broker specific transaction modifies the Hashtable by adding result fields. The server has a set of request object classes to allow it to act in different ways, for example, UBCBrokerRequest for general trading transactions that represents the bulk of the traffic, UBCSystemRequest for general statistics customized to the UBC implementation of the backend server, CommonRequest for common server oriented requests like stopping, starting and restarting the server.

Besides, the Messaging Servlet and Server also have a file-based configuration to allow flexibility and reuse.

Physically, each of the broker systems only need to connect to its own ISP, and access the 99stock system by VPN formed between the broker system via Internet. The TIBCO server is located at the 99stock system side, and the TIBCO client is located at the broker system side, all the requests and the returned results are transmitted on the virtual TIBCO bus formed between the TIBCO server and TIBCO client.

Fig. 4 illustrates further architecture of the Processing and Adapter layer in multi-broker networking system that forms one of the vital parts of the online trading system in Fig.3. In the Processing and Adapter layer of fig.4, there are Messaging layer interface, Broker Request Processing method and Java API and Broker API which form a hierarchy channel that conducts the information from/to Messaging layer to/from the end broker.

A broker API is a set of function calls provided for programming the brokerage trading system. The API provides the function calls to the broker functionality existing in the broker trading system. Those functionality includes placing an order, requesting an account or portfolio information, and so forth. For example, the functions in a broker API can be grouped into three parts. The first part is login/logout to the SQL Server. This is a pair that should be used at the beginning and at the end of the process inside a program. The second part is for trading activity processing. This part contains all the functions for placing orders, checking pending orders, and etc. The third part is fund related.

If the broker API is written in C++ and to make the API available in Java environment, it is needed to convert the API into Java API. In other word, one can access the API in Java. The Java language feature, JNI, will do this.

Fig. 5 illustrates the API architecture of the Processing and Adapter layer at broker side. In fig. 5, at bottom level is Uniform Processing Functions which process all the requests and returns the results to the Messaging Layer which routes the results based on the return values it gets. For example, the following eight basic request categories needed be included in Uniform Processing Functions to meet trading requirements are listed below:

| | |
|---|---|
| 1. | User verification |
| 2. | Place order |
| 3. | Cancel order |
| 4. | Portfolio summary |
| 5. | Daily transactions report |
| 6. | Cleared transactions report |
| 7. | Account history report |
| 8. | Change password |

For the above eight request types, eight classes are designed and implemented. They are UserVerification, Order, CancelOrder, Portfolio, TransDaily, TransClear, AccountHistory and Password. These eight classes will fulfil the tasks that are specified in the business specification.

In Fig 5, a Broker API is at the top layer. As described above, a broker API is a set of function calls provided for programming the brokerage trading system. The API provides the function calls to the broker functionality existing in the broker trading system. Those functionality includes placing an order, requesting an account or portfolio information, and so forth. The middle layer is C API and Java API. Owing to the Java language feature,

JNI, if the broker API is written in C, C++ and to make the API available in Java environment, it is needed to convert the API into Java API by C API. In other word, one can access the Broker API in Java through Java API and C API.

C Layer is a C wrapper of a C++ implementation. Within the C++ implementation the API is called.

For example:

```
extern "C"
JNIEXPORT void JNICALL Java_JieYiNative_Functions
  (JNIEnv* env, jclass cl)
{
      passing parameters
      return Cpp_JieYi_Functions;
}
```

Java Layer is a standard JNI to C layer, it needs to do works on the data formatting and converting.

Fig. 6 illustrates the interface to messaging layer at a broker system side. The interface to messaging layer i.e., Messaging / Processing Layer Interface, functions as the UBC Messaging architecture interface and instructions on how to use it to integrate with individual brokers. The implementation of the interface performs functionality mapping (Broker object Implementation) and message translations (Transaction Implementation). It will then run one or more of the broker's API functions to achieve the base set of request types defined by the broker. For example, in Fig.6, the interface links the Thread C to the Order function in a Broker API and connects Thread A, B to other corresponding Broker APIs.

This interface is the area that talks to the Messaging Layer. The purpose is to provide broker-processing classes to the Messaging Layer through an interface called *UBCTransaction*. This interface will contain a method called *process()*, which will take two parameters. One is a Hashtable and the other is a Broker object. The Broker object is required by the Messaging Layer to establish initialization of a processing thread.

```
public interface UBCTransaction {
  public Hashtable process (Hashtable h, Broker broker);
}
```

The Messaging Layer should use Broker object to initiate a processing thread and to clean up a processing thread when it is terminated. The definition of the Broker class is shown as followed.

```
public class Broker {
 public void start(){}
 public void end(){} }
```

The functions *start* and *end* will call the corresponding Java functions of *loginSqlServer* and *logoutSqlServer*.

As described above, the Messaging Layer requires that the input hashtable will minimally contain the entries of CustomerID, Password, BrokerID, BranchID and RequestType. The output hashtable will minimally contain the entries of the above plus Report and Error.

It is obvious that the input hashtable need to have input data in order to carry out a request. For example, an *Order* has to be accompanied with order type (buy or sell), number of shares, price and so forth. Detailed information will

be provided in the following sections. To include enough data, a new key *RequestData* will be added into the Hashtable. Also, the *Report* and *Error* will be added to input hashtable too. Thus, it is able to fix the format of a hashtable.

| ***Key Names*** | Descriptions |
|---|---|
| CustomerID | Investor Identification |
| BrokerID | Broker Identification |
| BranchID | Branch Identification |
| Password | Account Password |
| RequestType | Request Type, like Order |
| RequestData | Hashtable Contains Request Data |
| Report | Vector of Hashtable Containing Return Results |
| Error | Error Code and Message (ErrCode/ErrMsg). |

Broker Requests Processing Functions are a part that processes all the requests and returns the results to the Messaging Layer. The Messaging Layer routes the results based on the return values it gets.

Integrating with the Messaging Layer may require two activities: implementing the initialization that will span a plurality of transactions, implementing the transaction type functionality.

The Initialisation process in the Messaging Server comprises:

- Create a Message Queue. All incoming, TIBCO, messages will be queue whether there are available Threads or not. Each Thread will only access messages via the Message Queue.

- Create an ad-hoc queue handling Thread pool that starts empty. This pool will be used to process the Message Queue overflow. The pool should be internal to the Message Queue class.
- Create a cleaner Thread to monitor and maintain the processor pool. If a Thread signals that it is about to terminate then the cleaner Thread should start a new processor Thread. This Thread may later be extended to service information requests potentially via SNMP.
- Establish n Threads according to a CommonProcessorThreadCount environment variable. Each processor Thread will be of class UBCProcessor that extends com.ims.tibco.ProcessorThread class. Each UBCProcessor should instantiate com.ims.ubc.<UBCBrokerID>.<UBCBranchID>.Broker. The broker-specific class "Broker" will perform any one-time initialization and de-initialization that a particular broker requires. It also provides a state vehicle to use across a plurality of uses of a processor Thread. An example of this is establish a broker DBMS connectivity with the broker supplied API. If the instantiation of the Broker class throws an exception then terminate the thread. If instantiation is successful, each Thread should then enter a loop retrieving the next message from the Message Queue and then process it with the broker specific class. The retrieval process should block, waiting for a message to enter the Message Queue.
- Start TIBCO listener Thread. The listener thread should push the message onto the Message Queue. Before the message is added to the queue, the Message Queue should check to ensure the new queue size does not exceed the CommonQueueMaxSize. If CommonQueueMaxSize is about to be exceeded then, the current message, and all incoming TIBCO messages from that point on, should be immediately replied to with a

Broker Connect Failure message. All future incoming messages will continually be ignored until the Message Queue size drops to CommonQueueMinSize. The reply TIBCO message will return a SubscriberFailure object to the original publisher, which can be handled in a customized way by the publisher. In normal circumstances, the publisher should expect a BrokerRequest object.

The process following the Initialization in the Messaging Server may includes:

- When a UBC Processor receives a message, it unpacks it and examines the object. It will be expected to be of the class Hashtable. According to the Hashtable values contained in the BrokerID, BranchID and RequestType keys, the specific, transaction processor class will be determined. Example: COM.IMSN.CITIC.Branch1.Portfolio will be the class that will be invoked using the name/values: RequestType=Portfolio, BrokerID=CITIC, and BranchID=Branch1. If the structure of all CITIC Branches is the same, the BranchID value can be considered as a branch type ID.
- Each Thread should track the number of messages it has processed and when it finishes processing the message that is the number equaling CommonThreadLife it should terminate.
- All broker specific classes should adhere to the Messaging API by implementing the UBCTransaction interface.

The design and implementation of a particular transaction's function for a specific broker is essentially unbounded. The only requirements enforced are:

- Each transaction class extends the Transaction Java interface.
- Each transaction class is in the package com.ims.ubc.<BrokerID>.<BranchID> according to the BrokerID and BranchID in the BrokerRequest.
- Each transaction class name is identical to the class name passed across in the RequestType key of the BrokerRequest.
- Each transaction will be expected to pass back the resulting variables as name / value pairs in the BrokerRequest object.
- A transaction that produces a plurality of rows of data will be expect to deposit the rows as a Vector under the key "Report". The Report Vector will be a Vector of Hashtable where the Hashtable key is the column name and the Hashtable value is the data element.
- All errors from the transaction level should be placed in the "Error" key. The sheer presence of the Error key suggests a failure of some sort. The front end-code, broker specific code will be expected to understand how to deal with the particular message or error code.

Next, with reference to Fig 7 and FIG 8, as exemplary embodiments, an explanation will be given to how UBC end to end solution is operated with the Messaging / Processing Layer Interface. Fig. 7 illustrates the related processing involved in the multi-broker networking system. Fig. 8 illustrates the processing involved in the servlet at the web server.

In Fig 7, once the Web Server receives a request in HTTP form from the Browser of a Web Client, the Server then publish it to brokerX.BranchY.request through the TIBCO server of the UBC Servlet in

the Web server(1) into TIBCO Bus. The UBC Server Dispatcher subscribe the request from TIBCO Bus to brokerX.BranchY.request(2), then pass an Object and Reply subject from the RV message to a Thread in the Thread Pool(3). The Thread performs a Broker specific backend processing(4), and then publishes the message to the brokerX.BranchY.request(5) into the TIBCO Bus through TIBCO Connectivity Wrapper. The Server receives the response that UBC servlet automatically subscribed to, and presents it to the Browser in HTTP form.

In Fig. 8, further details are given to the Web Server. In the Web Server, HTTP connectivity Servlet is operative to receive the request from the browser and send it to a Transaction Creator, and to receive the reply from the Presentation Layer and transfer it to the browser. From Fig 8, it will be seen that the Transaction Creator, the Presentation Layer and TIBCO Layer consist of UBC servlet. The Transaction Creator generalizes the request as previously discussed. The Presentation Layer reformats the reply data and The TIBCO layer publishes a Transaction request to and receives a reply of the Transaction request from TIBCO bus.

Now an explanation is given to an example of trading request life cycles in association with Fig 3 to Fig 8.

1) Web Browser connects with Web Server via Messaging Servlet i.e., UBC Servlet such as a TIBCO Server.
2) Messaging Servlet takes name/value pairs and formulates data into a BrokerRequest.

3) The Messaging Servlet establishes a TIBCO connectivity to publish the new request. A new connectivity with the TIB can be established each time.

4) The Messaging Servlet publishes the request to TIBCO using the subject **TRADING.<brokerID>.<branchID>.request** formed from the BrokerID and BranchID supplied in the request.

5) The Messaging Servlet subscribes to a temporary TIBCO subject, unique to the current publish, and waits for a reply.

6) The Messaging Server, i.e., UBC Server such as a TIBCO client, at the broker end listens on the TIB as a client for the subject **TRADING.<brokerID>.<branchID>.request**.

7) The Messaging Server receives the message from the TIB and unpacks it.

8) The Messaging Server takes the request information and passes it to a broker specific, custom transaction class for processing (Processing Layer).

9) JNI method implementations will translate the message into a form suitable for handing off to the broker's native API. JNI method implementations will usually be in C (Adapter Layer).

10) The native Broker API performs the business logic and returns the transaction result.

11) JNI methods returns information to Java realm (Processing Layer)

12) The Processing Layer packages the results into a BrokerRequest and passes it back to the Messaging Server Layer.

13) The Messaging Server publishes reply on the temporary subject provided in the original message.

14) The Messaging Servlet receives reply and passes to the message to the Presentation layer.

15) The Presentation layer formats the data and passes it to the client (Browser / applet / other).

Fig. 9 illustrates physical architecture of an embodiment of the present invention, wherein quote and other news and other service can be provided utilizing the same TIBCO P/S virtual bus. Such an architecture comprise: Client Browsers which accessing Internet, Internet as a transmission medium, IP director for directing each of the messages to a plurality of Web Servers connected thereto and for balancing the load of each of the web servers, Web Servers for receiving and outputting the corresponding message, Java-based server Name Server connected to Web Server for allowing only the Web Server traffic to reach Java-based servers for load balance, Java-based servers connected to Java-based server Name Server for performing different functions such as trading, Quote, News, 8 factors(stock selection) etc, and Broker VPN server networked to Java-based server through VPN via Internet for providing respective services.

In addition, there are two firewalls, one is an external firewall which defenses for the IP director, Web Servers, Java-based server Name Server and Java-based server, the other is an internal firewall which protects the Java-based server Name Server and Java-based server.

From this architecture, besides the real many-to-many solution achieved according to the present invention, one can see further advantages for using TIBCO publish/subscribe technology in the 99stock Internet online trading system according to the present invention. Other services, such as quote, news, or stock selection can be provided in the same virtual bus without any

further additional special configuration. It brings about the scalability and flexibility of the system.

Fig. 10 illustrates a configuration for sending information to a user pre-defined facilities, such as pager, fax or e-mail, according to a user pre-set criteria. In Fig. 10, many paths by which inform a user a quote are showed. Quote information is fed in TIBCO Quote Publisher from outside. Then, through the TIBCO Bus, such information is received by a Notification Server and thereby is distributed with the user profile information into fax server, E-mail server and paging server. After the quote and the user profile information are obtained, the public paging service server calls the expected user's pager through an airnet. Thus, a desired user can be notified the quote information he wants in different ways. The user can receive his trading results by means of the notification function. He can also set his criteria for selecting the information, and the relevant information in the virtual bus will be received according to the topic the user set, and transmitted to the user by the notification service.

## Industrial applicability

This invention provides an universal broker connectivity system and an Internet online trading system which can be networked to a plurality of broker systems, therefore provides a many-to-many solution which is an effective tool for end users to do online securities trading via internet at different broker systems. This universal broker connectivity system can be interfaced to the broker systems having different programming and different configurations by means of a uniform messaging/processing interface. It

utilizes the publish/subscribe technology in a bi-directional manner, and provides an advantageous hierarchic format of the request and processing results so as to identify the destination receiver of a certain information, thereby realizing online trading between a plurality of investors and end users and a plurality of broker systems. Consequently, this invention provides an infrastructure that will produce business functionality at the front-end independent of backend technology constraints and have a middle tier completely generic and useful to all functionality.

Furthermore, based on the teachings in this patent application, the multi-broker networking technology can also be applied to other fields, for example, the different existing credit card systems provided by different banks, and other service systems of a same category but are belonging to different service providers and may have somewhat different hardware or software configurations.

While the particular embodiments of the present invention have been described in detail above, it should not be conceived that the present invention is limited to them. On the contrary, it should be noted that many modifications and changes thereto may be made within the spirit and scope defined by the following claims of the present invention.

Claims

1. A multi-broker connectivity system for networking to a plurality of broker systems, said connectivity system having a plurality of users, and comprising:

a receiving means for receiving users' request and formatting the request into a predetermined format, said format including sender's ID and receiver's ID;

a messaging means for transmitting data in said predetermined format to a predetermined receiver according to said receiver's ID;

at least one processing/adapting means each adaptively interfaced to a specific broker system, for receiving request for the respective specific broker system so as to perform processing of a certain class on a received request, and returning the processing result in the same format as the corresponding request to said messaging means while roles of the sender and receiver are exchanged;

a presenting means for receiving said returned result and de-formatting said result to present the result to the corresponding request sender.

2. The system according to claim 1, wherein said receiving means is a HTTP connectivity unit resident in a web server for receiving requests input by the users with web browsers.

3. The system according to claim 2, wherein said messaging means includes a publish/subscribe server unit and a publish/subscribe client unit with a virtual bus formed therebetween.

4. The system according to claim 3, wherein said format when receiving request is a hashtable comprising hierarchic levels which includes user's ID, processing system's ID and request; and said format when returning result is a hashtable further comprising result besides said levels of the corresponding request hashtable.

5. The system according to claim 4, wherein said presenting means is a component in said web server for generating web pages and presenting web interface to users.

6 .The system according to Claim 5, wherein the p/s server unit comprises a servlet for packing request and sending it to the virtual bus and invoking presentation layer when results are received.

7. The system according to Claim 6, wherein the p/s client comprises a server for receiving request from the virtual bus, providing a queuing mechanism to handle over flow of incoming requests, providing a multithreaded processing environment fed by the queue, and invoking the appropriate processing class in the processing and adapter layer to the broker and request type.

8. The system according to Claim 7, wherein the processing and adapter means perform any processing-system-specified functionality and is implemented on a request type by request type basis.

9. The system according to Claim 8, wherein the processing and adapter means provides access to native processing system's API and performs

message translation to appropriate format for native processing system's API.

10. The system according to claim 9, wherein said processing and adapter means comprises a first API layer, a second API layer before calling native processing system's API to form a uniformed messaging/processing-adapting interface, wherein the input request calls the respective functions of the first API according to the request's type, the first API calls respective functions of the second API, and the second API calls respective processing functions of said native processing system's API.

11. The system according to claim 10, wherein said uniformed messaging/processing interface is Java interface, and wherein said first API is Java API, said second API is C API and said native processing system's API is in C/C++.

12. The system according to claim 11, wherein said processing/adapting means comprises request information translator, functional-specified processor, before calling said Java API.

13. The system according to claim 12, wherein said P/S server is at the web server side and the P/S client is at the broker system side.

14. An online trading system utilizing the multi-broker-connectivity system according to any one of claims 1-13, wherein said multi-broker-connectivity system is networked to each of the broker systems by means of VPN via Internet.

15. A multi-processing-system networking system for networking to a plurality of processing systems, said networking system having a plurality of users, and comprising:

a receiving means for receiving users' request and formatting the request into a predetermined format, said format including sender's ID and receiver's ID;

a messaging means for transmitting data in said predetermined format to a predetermined receiver according to said receiver's ID;

at least one processing/adapting means each adaptively interfaced to a specific processing system, for receiving request for the respective specific processing system so as to perform processing of a certain class on a received request, and returning the processing result in the same format as the corresponding request to said messaging means while roles of the sender and receiver are exchanged;

a presenting means for receiving said returned result and de-formatting said result to present the result to the corresponding request sender.

16. The system according to claim 15, wherein said receiving means is a web server for receiving requests input by the users with web browsers.

17. The system according to claim 16, wherein said messaging means includes a publish/subscribe server unit and a publish/subscribe client unit with a virtual bus formed therebetween.

18. The system according to claim 17, wherein said format when receiving request is a hashtable comprising hierarchic levels which includes user's ID, processing system's ID and request; and said format when returning result is

a hashtable further comprising result besides said levels of the corresponding request hashtable.

19. The system according to Claim 8, wherein the processing and adapter means performs any processing-system-specified functionality and is implemented on a request type by request type basis.

20. The system according to Claim 19, wherein the processing and adapter means provides access to native processing system's API and performs message translation to appropriate format for native processing system's API.

21. The system according to claim 20, wherein said processing and adapter means comprises a first API layer, a second API layer before calling native processing system's API to form a uniformed messaging/processing-adapting interface, wherein the input request calls the respective functions of the first API according to the request's type, the first API calls respective functions of the second API, and the second API calls respective processing functions of said native processing system's API.

22. The system according to claim 21, wherein said uniformed messaging/processing interface is Java interface, and wherein said first API is Java API, said second API is C API and said native processing system's API is in C/C++.

23. A method for networking users to a plurality of processing systems, comprising the steps of:

receiving user's request and formatting the request into a predetermined format, said format including sender's ID and receiver's ID;

transmitting said request in said predetermined format to a predetermined receiver according to said receiver's ID;

receiving said request in said predetermined format for the respective specific processing system and performing processing of a certain class on the received request,

transmitting said processing result in the same format as the corresponding request from said receiver to said sender according to said sender's ID;

de-formatting said return result to present the result to the corresponding request sender.

24. The method according to claim 23, wherein said processing systems are broker systems.

25. The method according to claim 23 or 24, wherein said request transmitting step includes publishing said request in said format on a publish/subscribe virtual bus.

26. The method according to claim 25, wherein said result transmitting step includes publishing said result in a same format on the virtual bus.

27. The method according to claim 26, wherein said format when receiving request is a hashtable comprising hierarchic levels which includes user's ID, processing system's ID and request; and said format when returning result is a hashtable further comprising result besides said levels of the corresponding request hashtable.

28. The method according to claim 27, wherein said presenting step includes generating web pages and presenting web interface to users.

29. The method according to Claim 28, wherein said receiving and processing step comprises the steps of receiving request from the virtual bus, providing a queuing mechanism to handle over flow of incoming requests, providing a multithreaded processing environment fed by the queue, and invoking the appropriate processing class in the processing and adapter layer to the broker and request type.

30. The method according to Claim 29, wherein the processing/adapting steps comprises the steps of providing access to native processing system's API and performing message translation to appropriate format for native processing system's API.

31. The method according to claim 30, wherein said processing/adapting steps comprises the steps that the input request calls the respective functions of a first API layer according to the request's type, the first API layer calls respective functions of a second API layer, and the second API layer calls respective processing functions of said native processing system's API.

32. The method according to claim 31, wherein said first API is Java API, said second API is C API and said native processing system's API is in C/C++.

33. The method according to claim 23, wherein said processing/adapting step comprises the steps of translating the request information, functional-specified processing, before calling said Java API.

34. The method according to any one of claim 23 to 33, wherein said processing system is broker system.

35. A method for adapting a common connectivity system to a plurality of processing systems, each has a native API including a plurality of processing functions, comprising the steps of :

providing a uniformed adapting interface, a first API layer, a second API layer in sequence between the common connectivity system and each of the specific processing system;

inputting uniformed processing instructions from the common connectivity system and to a specific processing system;

calling respective functions of the first API according to each of the instructions' type;

calling respective functions of the second API according to said respective functions of the first API; and

calling respective processing functions of said native processing system's API according to said respective functions of the second API.

36. The method according to claim 35, wherein said first API is Java API, said second API is C API and said native processing system's API is in C/C++.

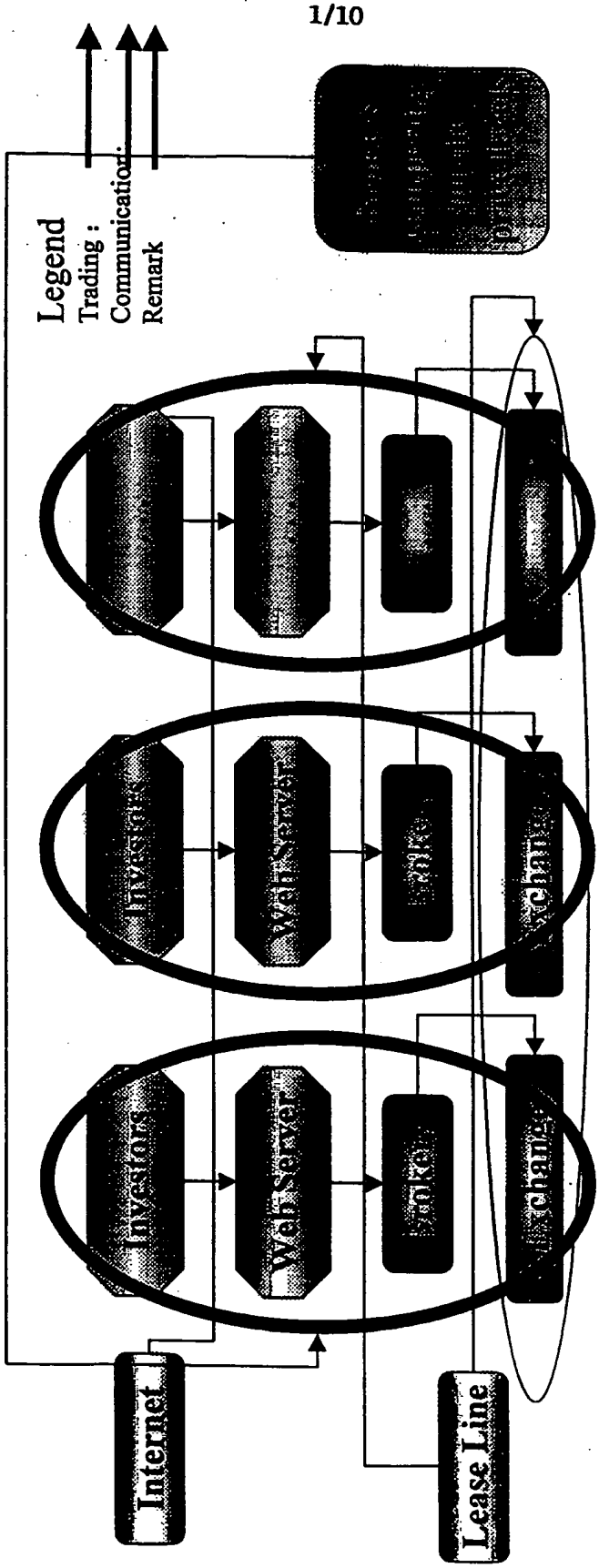



Fig. 1

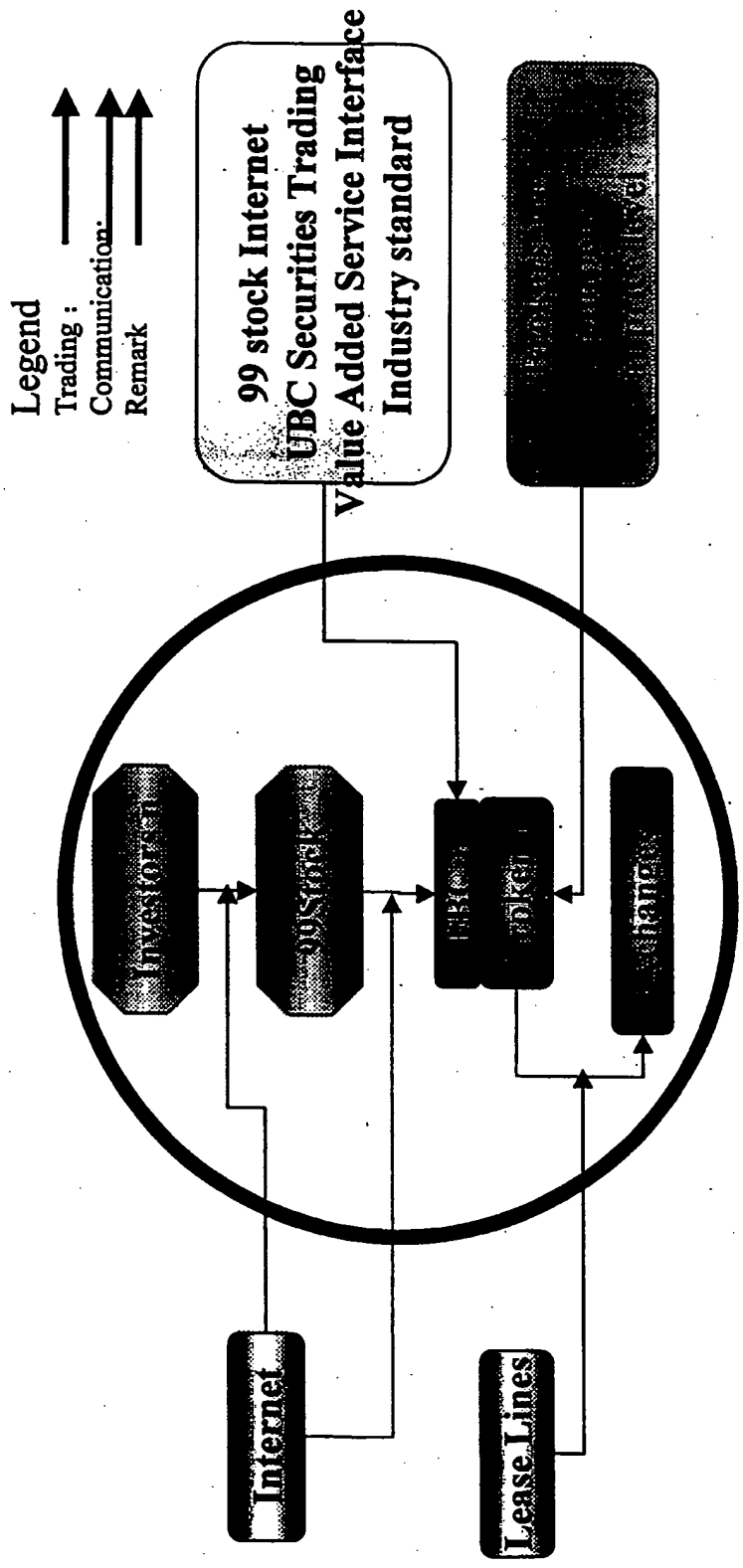

Legend
Trading :
Communication:
Remark
99 stock Internet
UBC Securities Trading
Value Added Service Interface
Industry standard
Internet
Lease Lines

Fig. 2a

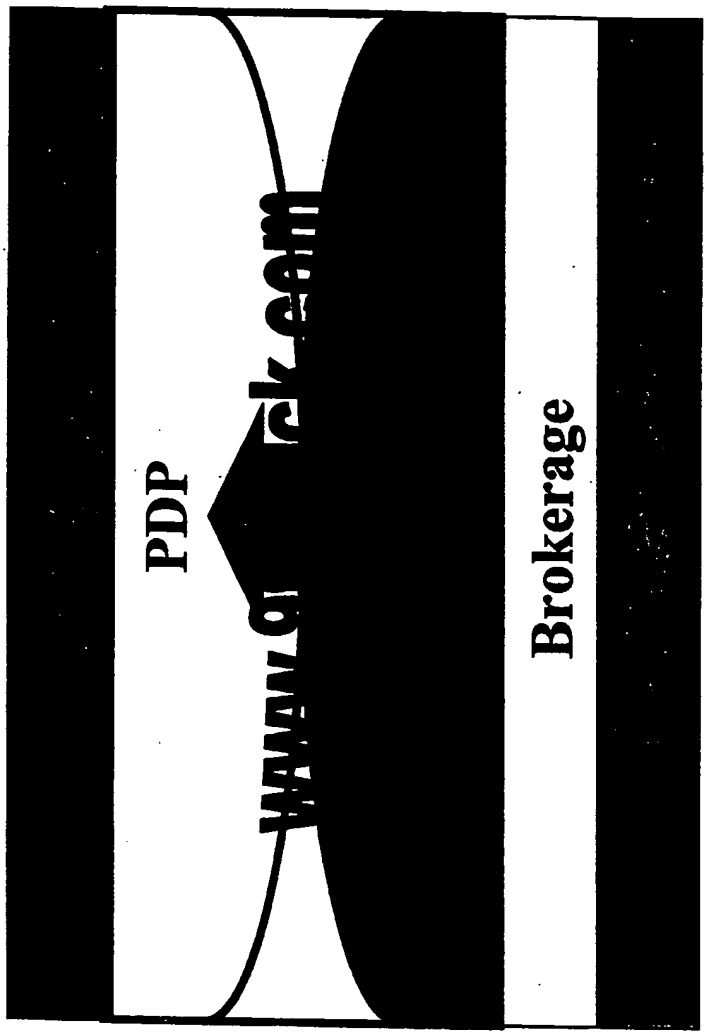



Fig. 2b

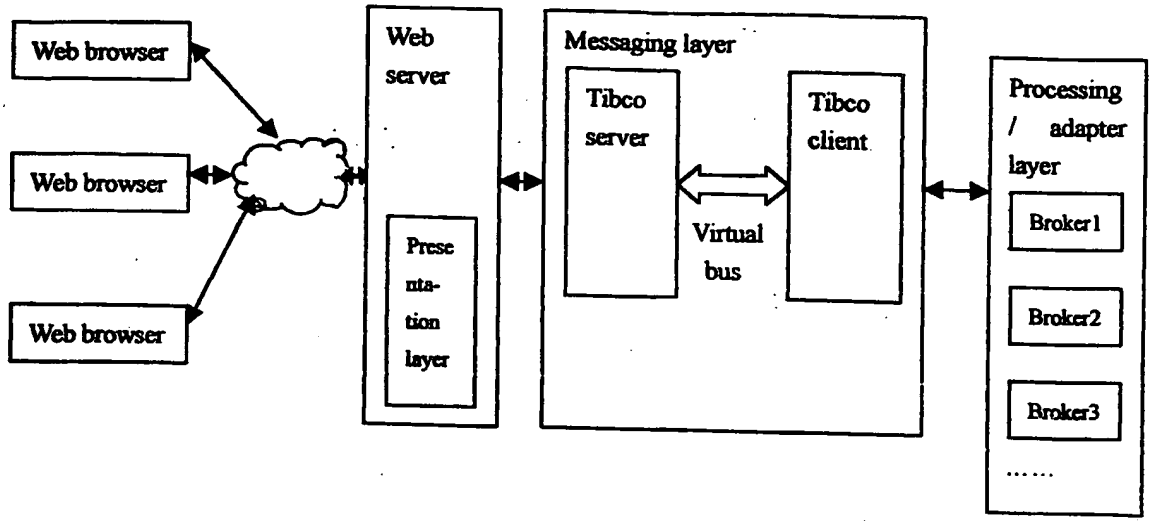

Web browser
Web browser
Web browser
Web server
Prese nta- tion layer
Messaging layer
Tibco server
Virtual bus
Tibco client
Processing / adapter layer
Broker1
Broker2
Broker3
.....

Fig. 3

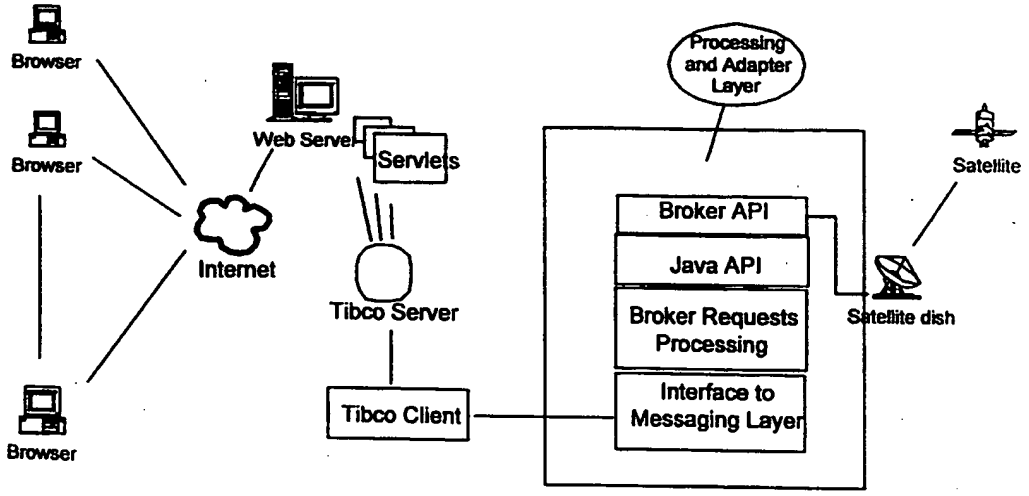

Browser
Browser
Browser
Web Server
Servlets
Internet
Tibco Server
Tibco Client
Processing and Adapter Layer
Broker API
Java API
Broker Requests Processing
Interface to Messaging Layer
Satellite
Satellite dish

Fig. 4

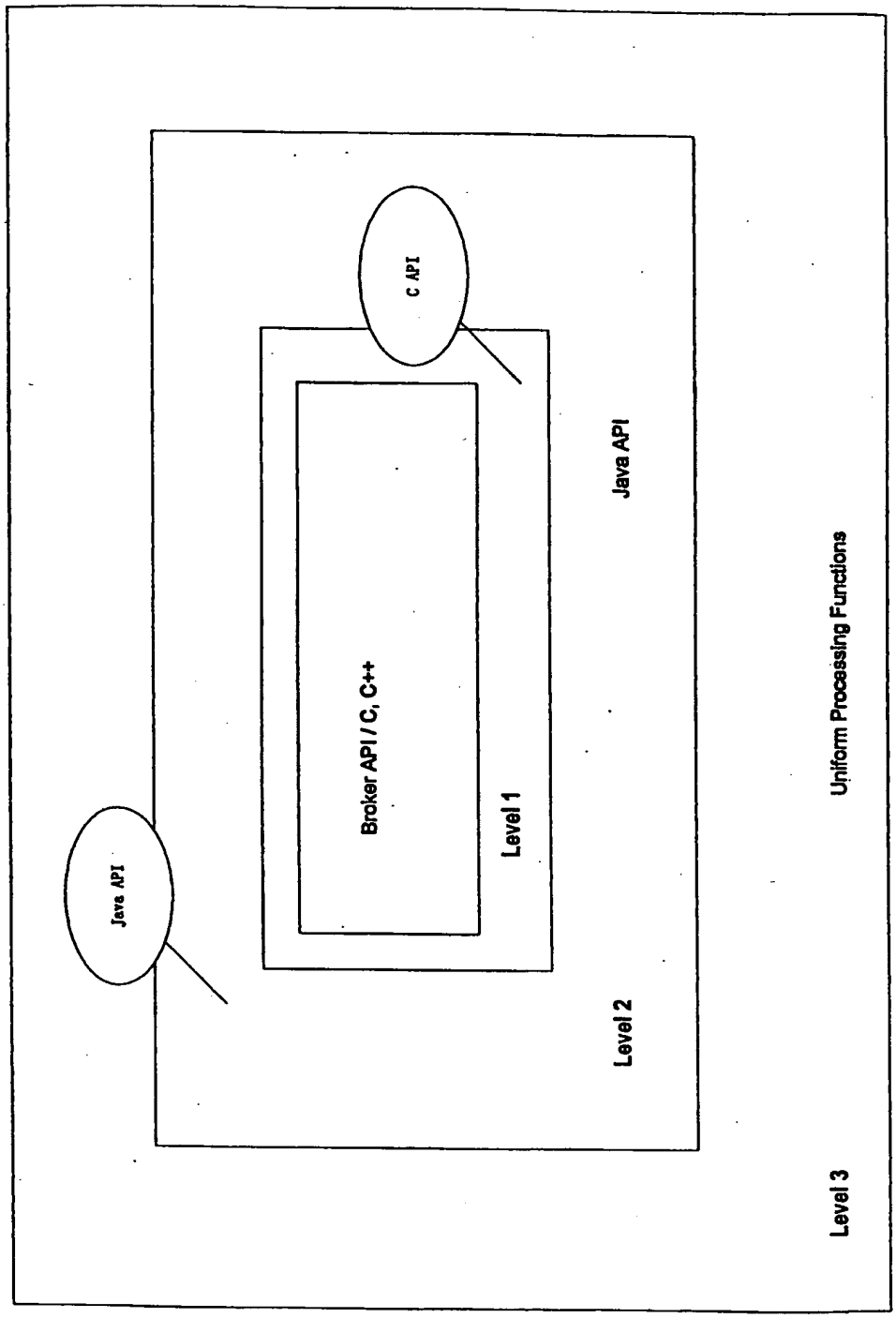

C API
Java API
Broker API / C, C++
Level 1
Java API
Level 2
Uniform Processing Functions
Level 3

Fig. 5

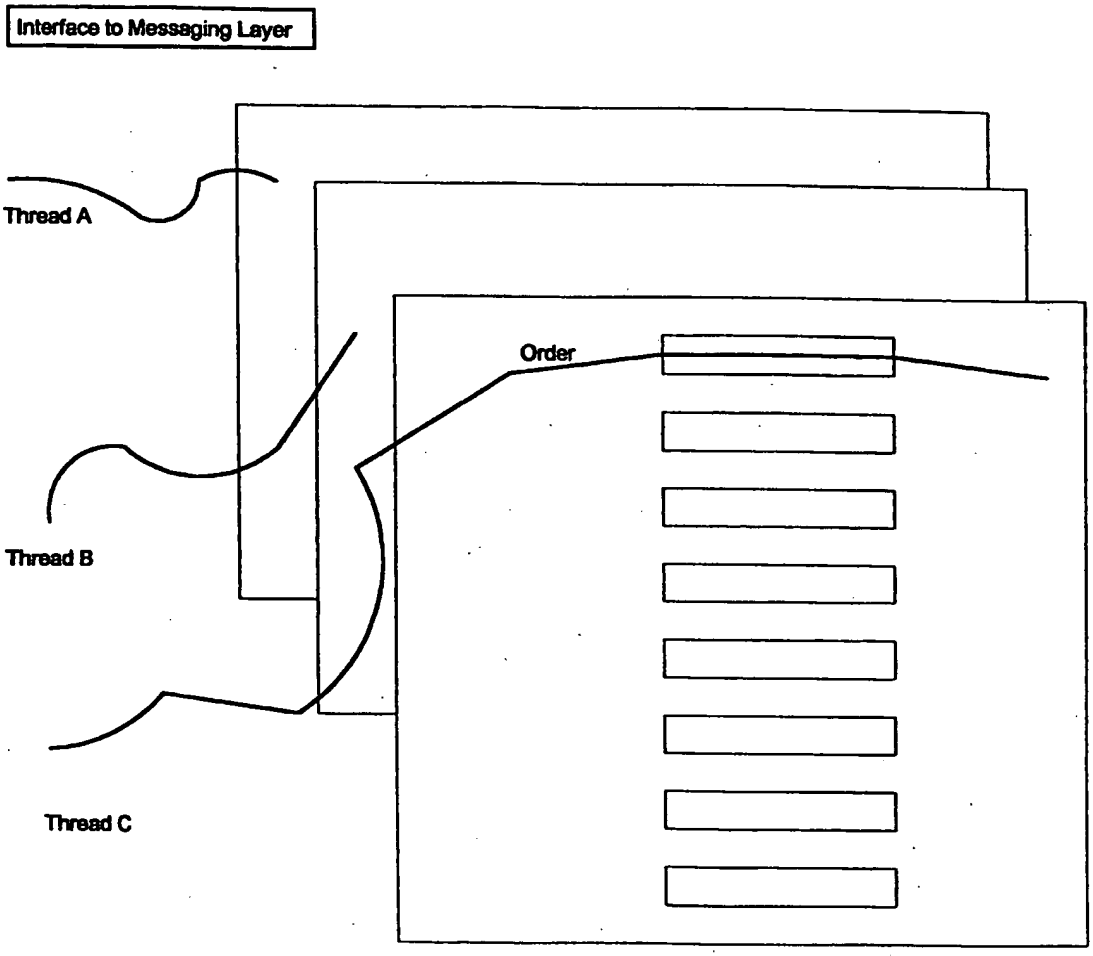

Interface to Messaging Layer
Thread A
Thread B
Thread C
Order

Fig. 6

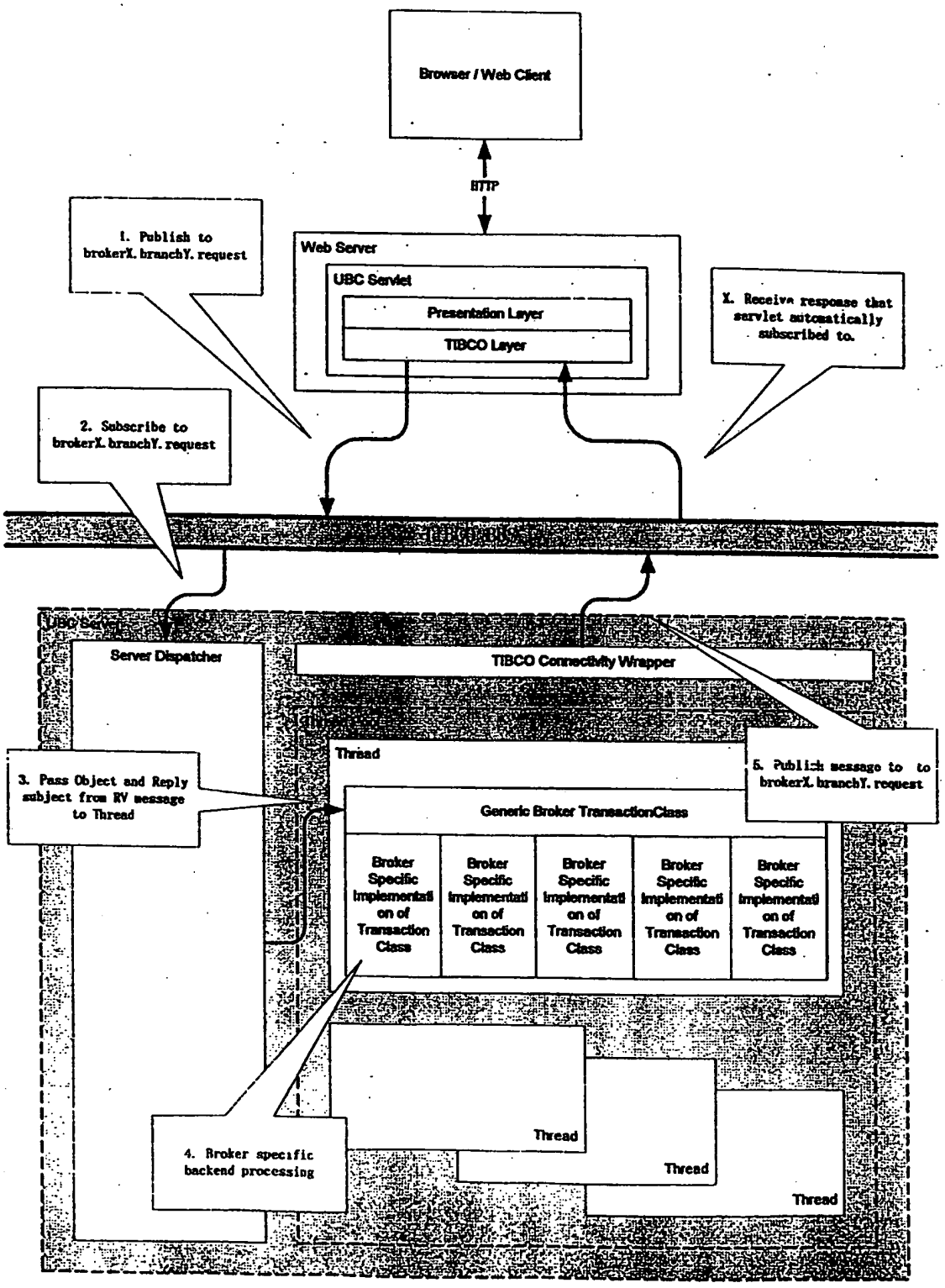

Browser / Web Client
HTTP
1. Publish to brokerX. branchY. request
Web Server
UBC Servlet
Presentation Layer
TIBCO Layer
X. Receive response that servlet automatically subscribed to.
2. Subscribe to brokerX. branchY. request
UBC Server
Server Dispatcher
TIBCO Connectivity Wrapper
Thread
5. Publish message to to brokerX. branchY. request
3. Pass Object and Reply subject from RV message to Thread
Generic Broker TransactionClass
Broker Specific Implementation of Transaction Class
Broker Specific Implementation of Transaction Class
Broker Specific Implementation of Transaction Class
Broker Specific Implementation of Transaction Class
Broker Specific Implementation of Transaction Class
Thread
Thread
Thread
4. Broker specific backend processing

Fig. 7

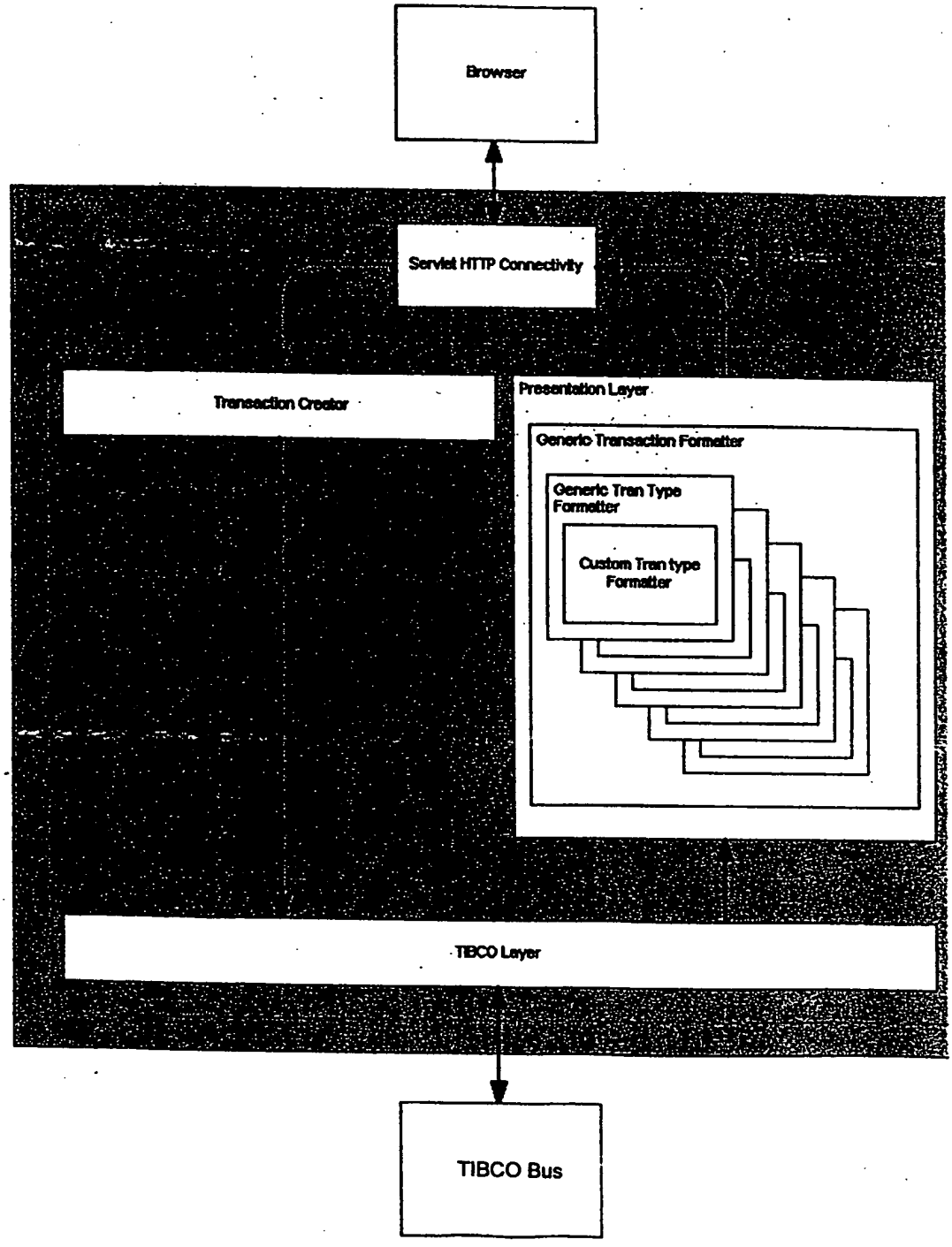

Browser
Servlet HTTP Connectivity
Transaction Creator
Presentation Layer
Generic Transaction Formatter
Generic Tran Type Formatter
Custom Tran type Formatter
TIBCO Layer
TIBCO Bus

Fig. 8

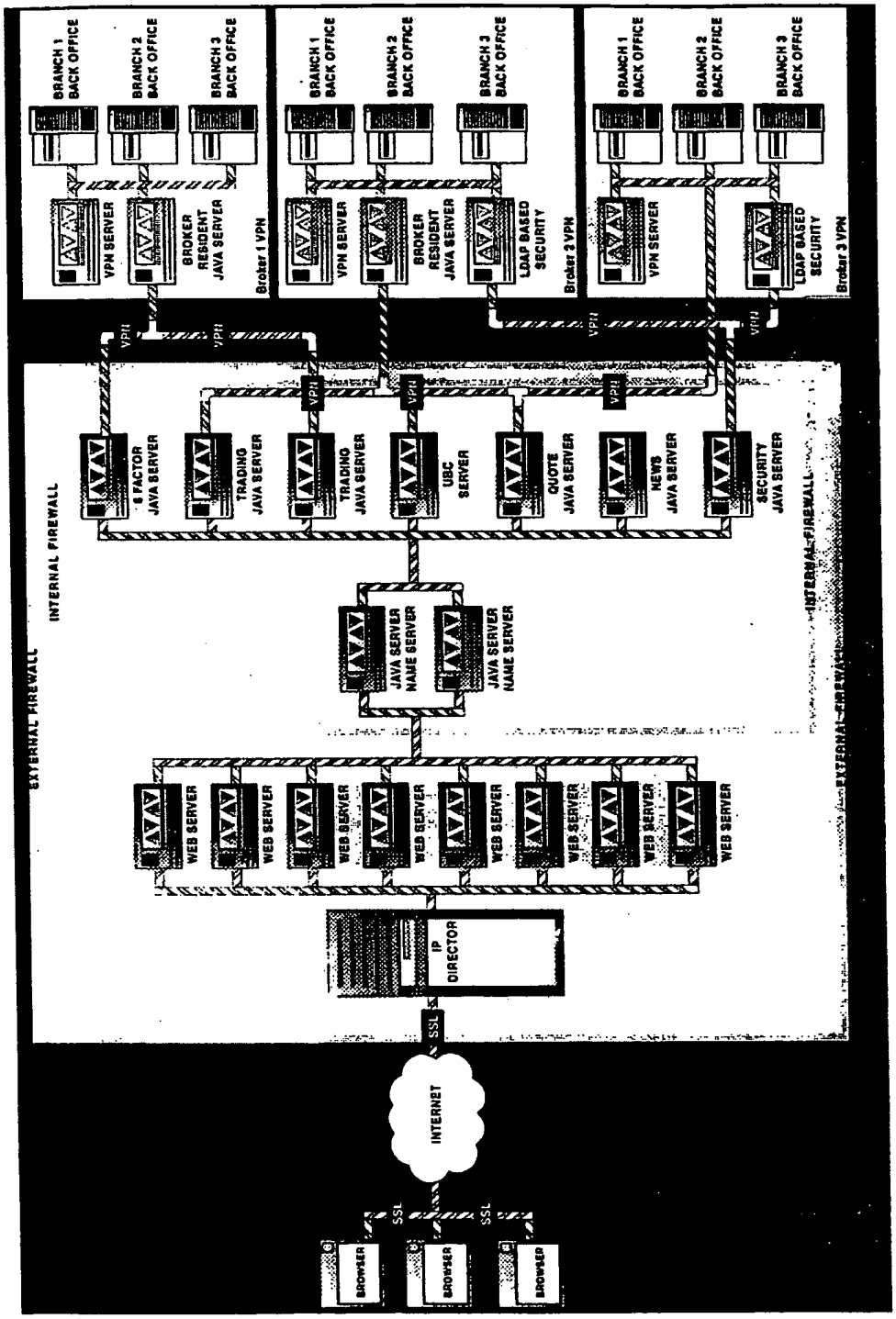

BRANCH 1 BACK OFFICE
BRANCH 2 BACK OFFICE
BRANCH 3 BACK OFFICE
VPN SERVER
BROKER RESIDENT JAVA SERVER
Broker 1 VPN
LDAP BASED SECURITY
Broker 2 VPN
Broker 3 VPN
VPN
EXTERNAL FIREWALL
INTERNAL FIREWALL
6 FACTOR JAVA SERVER
TRADING JAVA SERVER
UBC SERVER
QUOTE JAVA SERVER
NEWS JAVA SERVER
SECURITY JAVA SERVER
JAVA SERVER NAME SERVER
WEB SERVER
IP DIRECTOR
SSL
INTERNET
BROWSER

Fig. 9

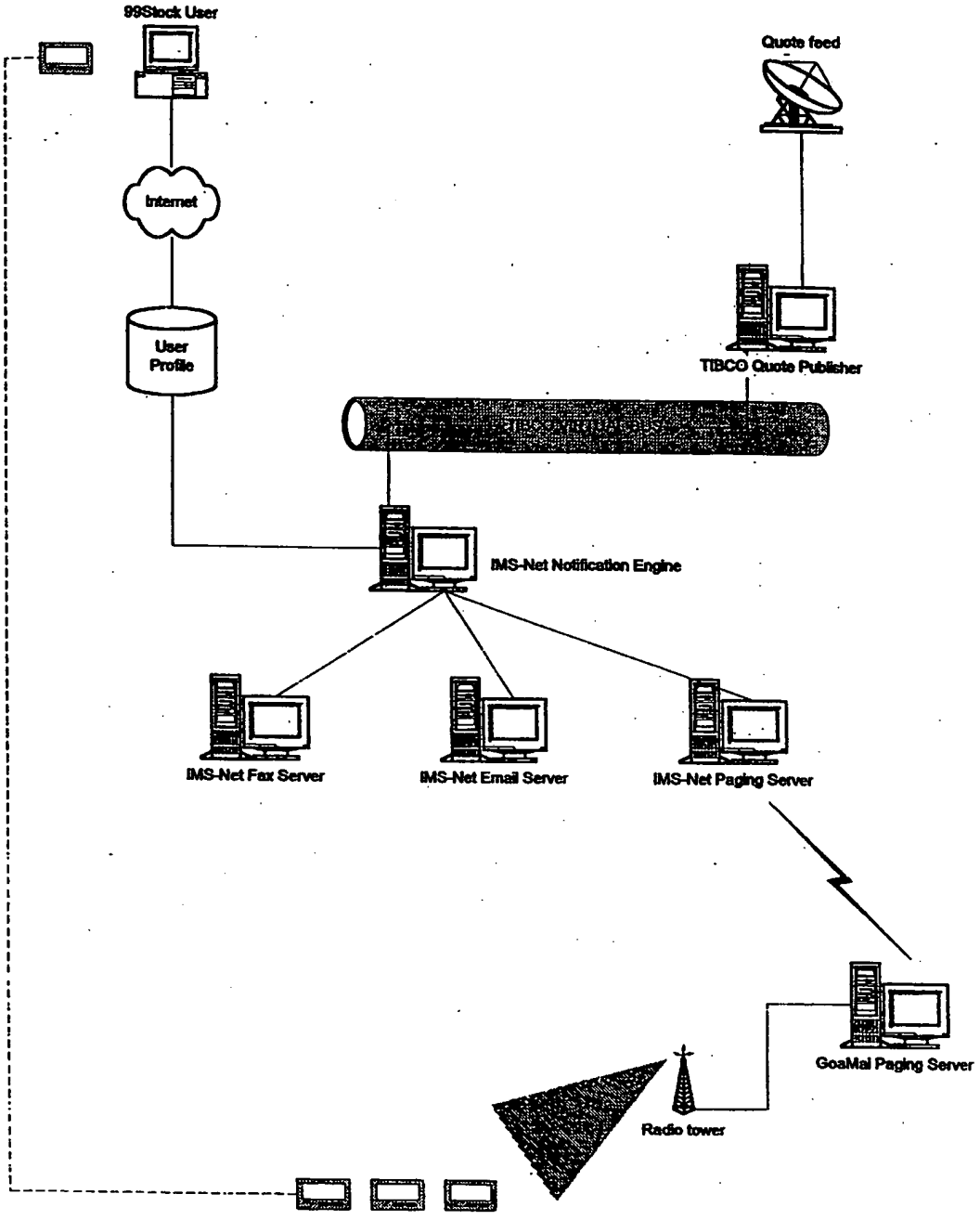

99Stock User
Internet
User Profile
Quote feed
TIBCO Quote Publisher
IMS-Net Notification Engine
IMS-Net Fax Server
IMS-Net Email Server
IMS-Net Paging Server
GoalMal Paging Server
Radio tower

Fig. 10

# INTERNATIONAL SEARCH REPORT

International application No.
PCT/CN 99/00031

A. CLASSIFICATION OF SUBJECT MATTER

G06F 17/60

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

G06F 17/60, 19/00, 15/20®, 15/21®, 15/22®, 15/24®, 15/16, 17/60, 15/00

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

C. DOCUMENTENTS CONSIDERED TO BE RELEVANT

| Category* | Citation of document, with indication, where appropriate, of the relevant passages | Relevant to claim No. |
|---|---|---|
| A | WO 96/21192A1 The whole document | 1—36 |
| A | WO 97/03423A1 The whole document | 1—36 |
| A | WO 97/36253A1 The whole document | 1—36 |
| A | WO 95/18418A1 The whole document | 1—36 |
| A | WO 91/14231A1 The whole document | 1—36 |

☐ Further documents are listed in the continuation of Box C. ☒ See patent family annex.

* Special categories of cited documents:
"A" document defining the general state of the art which is not considered to be of particular relevance
"E" earlier application or patent but published on or after the international filing date
"L" document which may throw doubts on priority claims(s) or which is cited to establish the publication date of another citation or other special reason (as specified)
"O" document referring to an oral disclosure, use, exhibition or other means
"P" document published prior to the international filing date but later than the priority date claimed
"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"&" document member of the same patent family

| Date of the actual completion of the international search | Date of mailing of the international search report |
|---|---|
| 08 June,1999(08. 06. 99) | 01 JUL 1999(01. 07. 99) |
| Name and mailing address of the ISA/CN<br>6 Xitucheng Rd. Jimen Bridge, Haidian District, 100088 Beijing, China<br>Facsimile No. 86-10-62019451 | Authorized officer<br>Guohong, Xia<br>Telephone No. (86-10)62093837 |

Form PCT/ISA/210(second sheet)(July 1998)